Communications and Tracking Expert Systems Study

Preface

This research was conducted under the auspices of the Research Institute for Computing and Information Systems by T.F. Leibfried, Associate Professor of Computer Science, Terry Feagin, Professor of Computer Science, and David Overland, Research Associate, all at the University of Houston-Clear Lake.

Funding has been provided by the Tracking and Communications Division, within the Engineering Directorate, NASA/JSC through Cooperative Agreement NCC 9-16 between NASA Johnson Space Center and the University of Houston-Clear Lake. The NASA Technical Monitor for this activity is Oron Schmidt, Systems Techniques Section, Communications Performance and Integration Branch, NASA/JSC.

The views and conclusions contained in this report are those of the authors and should not be interpreted as representative of the official policies, either express or implied, of NASA or the United States Government.

INTERIM REPORT

UNIVERSITY OF HOUSTON-CLEAR LAKE

RESEARCH INSTITUTE FOR THE COMPUTING

AND

INFORMATION SCIENCES

RESEARCH ACTIVITY AI-1

COMMUNICATIONS AND TRACKING EXPERT SYSTEMS STUDY

Prepared by:

T. F. Leibfried, Jr., Ph.D. -- Principal Investigator
T. Feagin, Ph.D. -- Research Area Director
D. Overland, B.S.M.E. -- Research Associate

In cooperation with:
O. Schmidt -- Technical Monitor/NASA-JSC
30 January 1987

University of Houston-Clear Lake Research Activity AI-1
Communications and Tracking Expert Systems Study
Interim Report for Semester Period Ending 31 Dec 1986

EXECUTIVE SUMMARY

The original objectives of the study consisted of five broad areas of investigation:

1. Criteria and issues for explanation of C & T system anomaly detection, isolation, and recovery;

2. Data storage simplification issues for fault detection expert systems;

3. Data selection procedures for decision tree pruning and optimization to enhance the abstraction of pertinent information for clear explanations;

4. Criteria for establishing levels of explanation suited to needs;

5. Analysis of expert system interaction and modularization.

Progress was made in areas 1, 2, 3 and 5, but to a lesser extent in area 4 during Phase 1.

Among the types of expert systems studied were those related to anomaly or fault detection, isolation and recovery. Specifically, the interim results Harris and TRW T&C expert system studies were examined and work supplementing them with explanation facilities was initiated.

An expert system which is rule based may be thought of as a sequence of if-(condition)-then-(action and fact(s)) statements in an endless repeating loop. A given statement or rule, when its condition has been satisfied and it executes, is said to have "fired". The rule usually asserts "facts" which may satisfy the condition(s) of other if-statements or rules which in turn can assert actions and/or other facts and so on. The beauty of these systems is that they are flexible, easily expanded or modified, succinct, and readily understood by humans. Their problems are a general lack of structure, modularity for groups of related rules, and therefore poor maintainability. Also, in complex rule-based systems there are problems controlling the order in which the rules are applied. That is, a "resolution strategy" must be provided. Most expert systems of this type are each essentially one big program where object-oriented design and information hiding are relatively absent. These latter concepts are essential for data integrity of software systems. This is a major issue for any project as big as the Space Station, where the total software system size may be measured in millions of lines of code. The potential benefits of expert systems are too great to reject them out of hand, especially since the modularity issue is probably tractable. Modularity in such systems is being investigated as part of area number 5 above.

Results to date indicate that modularity is possible especially in the case of predictable expert systems (i.e., systems where rules cannot make new rules). In fact, virtually any predictable expert system is capable of being rewritten in a procedural language. In some case that may be even desirable for part, if not all, of a given system. This was done in this study for a simple subsystem based upon a system simulator written by TRW. The simulator, written in C, was rewritten in Ada. Then, fault diagnosis and explanation facilities were implemented to investigate area number 1 (explanation facility) and area number 5 (interaction and modularization) described above. The source code for this system is in Appendix B as MAIN_SIM_MOD3.ADA. Results of this portion of the the study are shown in item D of the body of this report. Briefly, the results indicate that, an explanation facility is best structured as an integral part of the system rather than as an appendage. Object-oriented modularization promotes data integrity, and that the capability of retrieving and using old data is probably best achieved through a procedural language. Object-oriented design was first suggested by Parnas and implies that each object (e.g., variable) should be controlled by one module and the resources necessary to modify or change any of its characteristics should reside only within that module.

The areas number 1 (expert explanation), number 2 (storage simplification) and number 3 (decision tree issues) were investigated by the implementation of a simple engine diagnosis program. The source code and concomitant comments for DIAG4.ART are in Appendix A. The results are shown in item B of the body of the report. Briefly, the results show that rules have a taxonomy (e.g., explanation rules, bookkeeping rules, action rules), and that the time stamping of facts is necessary for explanations of any past expert system actions. The need for functional or domain partitioning was one of the discoveries of this investigation.

Not much implementation of techniques for pruning decision trees of irrelevant decision nodes (area number 3) was accomplished; however, these and other data compression concepts were discussed extensively and some paths of investigation and experimentation are indicated. So-called spine optimization of the decision tree may provide a capability for explaining in retrospect how and why a decision has been reached or even perhaps why a particular decision was not reached. (A spine is defined as the conjunction of decisions for which the retrieval is a single conclusion.)

In the relatively short time this study has been in progress, what were originally nebulous issues have become more clear. This is not to say that they have become more tractable, but at least some of the issues are better defined than they were four months ago. We expect that this trend will continue.

BODY OF THE REPORT

A. Introduction

The original statement of activity objectives consisted of several items summarized as follows:

1. Examination of existing C&T systems configuration and monitoring problems--
   The activity started with an examination of fault detection expert systems and historical database storage, which is one facet of the subject area of C&T system anomolous behaviour. An anomolous event may be defined as the situation when normal sytems software has been unable to operate according to plan. Some tentative conclusions for structuring these activities have been determined. This activity continues.

2. Audit trail simplification policies--
   Some experimental work has been accomplished for this activity. In particular, audit trails for fault detection expert systems have been analyzed. An hypothesis has been formulated, based on preliminary results which indicates that shallow explanations may be able to use a limited audit database but deeper explanation facilities may require a parallel expert system, otherwise the audit trail database might become too cumbersome. The question of the effect of utilizing distributed expert systems on practical explanation facilities has not yet been considered, but that is a subject which will need to be addressed once the resources needed by such facilities have been identified.

3. Data selection procedures--
   This activity is closely related to item 2 but has been modified to focus upon decision tree pruning and storage depth of unchosen paths. For explanation purposes one hypothesis to be explored is to store chosen path nodes and one node level below each chosen path node for all unchosen paths. This would, at a minimum, double the storage required for a chosen decision tree but it would obviate the necessity for reexercising a duplicate of parts of the original expert system for shallow explanations. No general rule with specifics has yet been established but some simple subsystem simulations, namely, DIAG4.ART and MAIN_SIM_MOD3.ADA, have given some insight into this problem. This activity continues.

4. Criteria development for user-expert system interfaces--
   This activity has been modified to focus upon identifying levels of explanation suited to various user's requirements. It will be assumed that user requirements will have already been identified. It will be assumed that short explanations will be supplied initially and that levels of depth may be requested at any point in the process. This activity has not seen much progress to date but will continue in the revised direction.

5. Expert system interaction--
Intercommunication and hierarchical decision priorities have been discussed but no definitive assertions have yet been determined. At first sight this issue seems related to partitioning and modularizing expert systems. This latter issue is one which is to be explored in the next phase of this study.

B. Explanation Facilities and Decision Trees

An explanation facility will be an important part of any system that is used to provide diagnostics for problems that may develop in the tracking and communication systems on the space station. Also, once the most probable source of a problem has been identified and corrective measures are begun, an explanation of why and how such actions have been taken will be necessary. It would appear that there are several levels of explanation that can be provided. At one extreme, a complete explanation with all the intricacies and details of all of the inferences used could be provided, such that the explanation would be effectively equivalent to providing a complete decision tree (in which only one path has been followed). At the other extreme, a brief, superficial explanation could be provided that would only indicate the most immediate reason why a particular path was selected. This might be of use in the case when a systems human monitor might have selected a different alternative than did the expert system and he seeks the reasons for the selection made by said expert system. In this latter case, it would be practical to store the path and the collection of decisions made along the path. However, simply quoting the path to the given conclusion will not, in general, provide a satisfactory explanation because there may be irrelevant decisions on the path and there may be a need for deeper explanations. It appears that the need for deeper explanations can only be satisfied if the complete reasoning process for reaching a given conclusion is available (i.e., either we provide access to all the rules that were used or we provide the complete decision tree). Such might be needed when a large scale manual intervention is planned for direct control of C&T systems and in that case irrelevant decisions would effectively be "noise" in the system. A procedure is available that would permit the removal of irrelevant decisions. It is called spine optimization, which consists of pruning the decision tree to obtain a "prime spine", and which prime spine is formed by delaying or removing irrelevant decision nodes in the tree. Most of the work so far on spines has been theoretical, so it is not clear that the benefits of such removal would justify the costs.

It may be, in some cases, that the option of providing the complete decision tree would not be practical in light of the large amount of storage that this would require for each of many points in the past. It is possible, however, that a few of the most recent decision processes could be stored in their entirety and that a number of older decision processes could be saved in an abbreviated form (perhaps with just a simple rundown on the decision path actually taken with only immediate explanations for taking a particular path provided).

If more elaborate explanations were required for older decisions, then it would still be possible, albeit time-consuming, to reload another knowledge base with the facts that were true at the time of interest and then begin to execute some of the explanation rules over again, only this time with the user interrupting that expert system and asking questions about the decisions which were made.

One of the dilemmas encountered with after-the-fact explanation facilities is that the explanation facility itself must provide some constructive filtering. Indeed, in order to remove irrelevant decision information the explanation program must prune and perhaps redo, albeit with a different perspective, some of what the diagnostic facility did in the first place. The simplest thing to do would be to supply the complete decision tree, but this would be barely one cut above the Automate Reasoning Tool (ART) dribble file as far as user utility is concerned.

An hypothesis or question which probably is worth considering is, "Should the explanation facility 'anticipate' queries so that it would effectively run in parallel to the diagnostic expert facility but without encumbering it?" Another possibility is to initiate the building of a more extensive event data base whenever an anomolous condition occurs.

At this point, without making a definite statement, let us offer a conjecture. We are of the opinion that storing the entire decision tree may be a viable solution, after all. It's not really as bad as storing the entire state at several points in time -- which might be needed since some problems will be building up over time -- in order to be able to provide explanations. The decision tree information would be kept active for a minimal time, presumably until time had passed when an explanation might be required, and then archived. We think that the decision tree also has just those rules used/needed to provide the explanation. If the other approach is used, we would have to ferret out those rules that were relevant or reload the entire expert system.

C. The ART Environment and the Attempt to Implement an Elementary Simulation, Diagnostic and Explanation System

The ART Environment

There are many advantages and some deficiencies to the Automated Reasoning Tool (ART) environment. The flexibility allows both forward and backward chaining with schema (which provides memory slots similar to frame-based languages), and viewpoints. The viewpoints may provide a way of recalling how a particular expert system operated at a given time in the past. This is something which would be useful for providing an explanation of a past event. Unfortunately, the first diagnostic and explanation example program did not use viewpoints and the lack of the availability of file input-output limited the scope of the program. Useful knowledge was obtained, nevertheless, and that should help the development of future explanation experiments.

Recommendations on Expert Systems Languages

A few observations have resulted from the first diagnostic-explanation system. The ART language coupled with the Symbolics system is a very versatile, albeit sometimes clumsy, approach to expert system development. The feasibility of developing an in-house (Ada-based) rule-based language should be investigated. This would allow data structures and I/O requirements to be tailored for the application.

This could also allow the development of subprocedure calls for both subprocedures written in sequential languages and those written in the development language (in other words, calling other expert system programs).

The DIAG4.ART Diagnostic-Explanation Program

Objectives

The goal of this program was to demonstrate at least a rudimentary explanation facility on a limited domain, mainly as a means of exploring the concepts involved, and also as a means of learning the ART language and Symbolics system.

Description

The program, written in ART, simulates the operation of a four-stroke, two-valve, single-piston internal-combustion engine. It also diagnoses failures in the ignition phase of the engine operation, and implements corrective action. It is then capable of explaining the diagnosis and the corrective action taken.

The program was also written so as to reflect some logical organization: The explanation rules are at the beginning of the program, followed by the bookkeeping rules (those responsible for updating the current parental and subgoals), followed by the initializing "split" rules, followed by the rules for action at each stroke of the cycle. The "split" rules were made necessary because the condition portion of a rule in ART does not have a provision to match on two OR'ed schema slots.

Program Outline

The engine state is modeled by a schema named CURRENT which is modified by the action of the stroke and ignition rules to reflect the operation of the engine. Each relevant component of the engine state is stored in a slot of the schema. At the same time, each slot in CURRENT is compared to a similar slot of the IDEAL engine state. Discrepancies, such as the spark plug not firing, cause error flags to be set which allow the diagnostic rules to fire. In this case, each flag triggers one diagnosis, but combinations of flags could also do this.

Diagnostic rules printout specific error messages and take corrective action (replacing the spark plug). They also query if an explanation is required. If one is, another flag is set.

The combination of the error and explanation flags allow the firing of explanation rules, of which there are two in this program.

Conclusions

The program served as a learning process, but ended up with serious deficiencies:

The fault tree was never more than two levels deep (both spark plugs fail) and did not demonstrate any combinatorial failures nor did it ever have to "guess". There couldn't be a wrong diagnosis.

All explanations were developed as the program "sequenced" through the simulation. No explanations could be given after the cycle continued: the program has no memory of what has already happened; it can only respond to the current state. This could be remedied by time-stamping all facts (saving the telemetry stream), but this would also require a new set of rules to react with past data in addition to those already existing which react with current data, effectively doubling the size of the program (at least).

All explanations either exist from the beginning or they could not be given. The drawback with programming in this style is that all possible faults and fault combinations must be figured in advance and coded into the program. This is trivial for a program this size, but is not practical for a large program. This increases the size of the program exponentially with the number of components. It also requires being able to simulate the "correct" functioning of the system at all times in order to detect discrepancies.

The source code for this program is shown in Appendix A.

Proposed Further Avenues of Exploration

The concept employed with this program would only allow explanations to be given "on the fly", that is, only after each diagnosis and/or fault correction. Explanation cannot be generated past that point in time; there is no memory of the transaction.

To avoid this fault then in order to generate explanations of past events either:

1. Save all explanations for possible future recall, or;
2. It must be possible to generate explanations from saved data.

There are a number of unexplored avenues here, such as the optimum way of generating explanations, the best way of storing data, etc.

Another concept to be explored is that of partitioning the expert system. The program, as currently written, does not support this, but the idea of breaking the system into modules, either functionally, where a module would perform certain tasks, or by domain, where each module would service a system or subsystem of the overall domain. The ability to do so would have quite an effect on the memory requirements and speed of the system.

D. Ada Implementation of a C&T Subsystem Simulator (TRW), Fault Detection and Explanation Facility with User Interaction

Motivation (objective)

This demonstration program was initiated for two reasons. First to demonstrate how a sequential software system could effectively duplicate the results of a simulation written in C and an expert system written in a specialized rule-based language such as ART (Automated Reasoning Tool by Inference Corp.). The second reason was to demonstrate that in contrast to conventional rule-based systems a sequential system can more easily store facts in a database and access them for an explanation facility. An ancillary reason was to examine how modularization could be used when implementing both diagnostic and explanation facilities.

Structure (program outline)

The program, written in Ada, consists of five modules called packages.

1. MAIN_SIM_MOD3

   This is really not a module but rather the driver program for the demonstration system. It essentially structures the system by calling subprograms. A simplified algorithm for this program is given in the following enumerated steps.

   (1) Initialize the communications system database;

   (2) Check for inconsistencies in the data, and if there are any, call a subprogram to ask the human monitor to correct the data;

   (3) Call the equipment emulator program;

   (4) Based upon the measured equipment output and the status of the switches, determine the condition of the equipment (i.e., diagnose the probable cause of failure, if any);

   (5) Call a subprogram to display the status of the equipment to the human monitor;

   (6) Call subprograms to provide an explanation of the diagnosis if requested;

   (7) Call a subprogram to interact with the human monitor to see if another simulation is to be run;

   (8) If the human monitor wishes to stop the program then terminate else call a subprogram to ask the monitor to update the simulation parameters and then go to step (2).

2. Y_OUT_MOD
   This package contains the equipment emulator subprogram. It effectively simulates the action of the hardware given the parameters in the equipment status database.

3. FAULT-ANALYZE
   This package contains the subprograms which measure the observable parameters and determines the probable fault, if any.

4. EXPLAIN_DIAGNOSIS
   This package presents an analysis of the reasons behind the fault diagnosis when requested by the human monitor.

5. I_O_SIM_MOD
   This package is the one which accesses and updates the simulated equipment parameters.

The source code for this system is shown in Appendix B.

Observations Based Upon Results

Not too much can be asserted with certainty but there are a few points which the work suggests. Among those are:

1. To explain even a moderately complex fault analysis decision it may be simpler to parallel a part of the decision process rather than try to filter the information from information written into the data base. Again, a trade-off exists between the classical performance parameters of execution time versus memory (primary and secondary storage). For example, in the diagnostic package called FAULT_ANALYZE the principal program DIAGNOSIS tests the output power EQUIP_Y_OUT_LEVEL and if it is less than -145 dbm it calls a "low level" program to examine the on-off switches in SWITCH_STATUS. If the main power switch is "ON" then the oscillator switches are tested. If the selected oscillator is "ON" then the program "reasons" that the selected oscillator is inoperative. Now when an explanation of this event is requested the EXPLAIN_DIAGNOSIS package is activated. Examining the code for procedure EXPLAIN_DIAGNOSIS we see that it parallels the reasoning of procedures DIAGNOSIS and SWITCH_STATUS in package FAULT_ANALYZE, that is to say, it has the same nested "if" structure. The only thing it adds is the diagnostic message, "We found the selected oscillator switch to be ON, the power output was in the noise level, so potentially we had a catastrophic oscillator failure." If the DIAGNOSIS and SWITCH_STATUS programs had been more cooperative they could have selected the appropriate literal value for an enumeration variable and stored this "hook" in the data base for access by the EXPLAIN_DIAGNOSIS package. Then all the EXPLAIN_DIAGNOSIS procedure would have to do is examine this variable by virtue of a simple "case" statement and supply the quoted explanation, "We found..etc."

Such a "type" definition for the desired variable might be:

```
type Switch_Permutation_Type is
     (POWER_OFF,
      POWER_ON_OSC_OFF,
      POWER_ON_OSC_ON);
```

A variable of this type could be set equal to one of the enumeration literals by procedure SWITCH_STATUS in package FAULT_ANALYZE.

2. Communicating with a user and writing facts to files and/or ephemeral data storage which facts are useful for explanation facilities may require interfacing a given expert system to procedural language I/O routines.

3. It seems that it should be possible to modularize a given expert system to some extent. For example, a FAULT_ANALYZE subsystem could in large measure be separate from an EXPLAIN_DIAGNOSIS subsystem. There would probably be some shared data and possibly even some shared utility routines, but the main thread of each subsystem could be separate.

4. One of the deficiencies of this implementation is that explanations are done with the same database as the fault analysis system, and before any recovery corrections are made. Thus, the database is unchanged when the explanations are made. The programming system could easily be altered to allow changes in the data and still allow an explanation after the fact. The technique employed would be to create two variables for each entity, one for the old value and one for the current value. This is easily done in Ada or in any procedural language but is more difficult in a so-called expert systems language. This may indicate that any production system would require its expert systems language to provide an interface to procedural language subprograms.

Future Direction

The next task which may be proposed is to expand this demonstration program to include more realistic fault detection with more realistic "hook" data generation. Then the explanation subsystem could be restructured to use these improvements. A parallel system in ART will also be implemented (if feasible).

SUMMARY OF OVERALL RESULTS AND RECOMMENDATIONS

A. The issue of storing the complete decision tree versus only storing the knowledge base for an explanation facility has not been resolved, but a few alternatives to be explored have been identified.

1. Store the decision tree in its entirety with perhaps one branch node for paths not taken at each node in the decision tree.

2. Build and store a complete knowledge data base whenever an anomaly occurs. (This could be any undesirable outcome such as the software system displaying unanticipated behavior. This could be signalled by the astronaut monitor or the expert system itself such as when a system failure is detected by the low level systems.)

3. Store only the facts with appropriate time stamps, and when an explanation is required, load a system containing rules similar to the original expert system so as to effectively parallel the operation of the original expert system but this time allowing the user to interrupt this parallel system to ask for appropriate explanations about paths not taken.

These alternatives are not mutually exclusive but all should be examined in future work.

B. The issue of the features of an expert systems language which language is appropriate for development and perhaps implementation of software systems which meet C & T functional requirements needs to be addressed. This recommendation is based upon the experience acquired by the structuring of a simple simulation and diagnostic-explanation program written in ART, (DIAG4.ART in Appendix A), the program had to be all in one module. In addition, there were the file I/O deficiencies of ART and the difficulty of storing more than one value for a given parameter. As a minimum any such language should be capable of interfacing with a compiler language program in a straightforward manner.

C. It is recommended that the issues of object-oriented design for expert systems be raised and investigated. In addition, the concept of supplying adequate "hooks" for explanation should be addressed early. Explanation facilities are best implemented when they are built in and not just "added" as a separate entity. If this is not done the explanation facility requires additional resources and must parallel some previously "paths" already trodden by other systems. The issues of object-oriented design, modularization, and so-called information hiding may not be just academic but a necessity for implementation of any large and perhaps distributed control and monitoring system, be it a procedural and/or rule-based software system. The topic of object-oriented design for expert systems addressing anomaly detection, recovery and explanation was examined by structuring a procedural software system in Ada. The system consisted of a driver and hardware status simulator, fault diagnosis, and fault explanation modules for a small radio frequency communication subsystem. Implementing expert system functions in Ada indicated that modularization and object-oriented design indeed are feasible without compromising effectiveness.

APPENDIX A

Source Code for DIAG4.ART

an Engine Diagnostic/Explanation Program

```
;;; -*- Mode: ART; Base: 10.; Package: ART-User -*-
(defschema engine-state "state of engine - beginning of intake-stroke"
           (carburation good)
           (piston-direction descending)
           (sparkplug1 good)
           (sparkplug2 standby))

(defschema current
           (instance-of engine-state))

(defschema ideal
           (instance-of engine-state))

(deffacts state
          (state-name intake))

(defrule compare-ok "compares current and ideal if same"
         (schema current (carburation  ?state3))
         (schema ideal   (carburation  ?state3))
         (schema current (piston-direction ?state4))
         (schema ideal   (piston-direction ?state4))
=>
         (printout t t "compared ok" t t))

(defrule compare-not-carburation
         (schema current (carburation  ?state3))
         (schema ideal   (carburation  ~?state3))
=>
         (printout t t "carburation compared not ok" t t)
         (assert (error-trap carburation)))

(defrule compare-not-direction
         (schema current (piston-direction ?state4))
         (schema ideal   (piston-direction ~?state4))
=>
         (printout t t "piston-direction compared not ok" t t)
         (assert (error-trap direction)))

(defrule no-ignition
         (state-name power)
         ?ignition <- (ignition fail)
=>
         (printout t t "sparkplug did not fire" t t)
         (printout t t ?ignition t t)
         (assert (error-trap ignition))
         (retract ?ignition))

(defrule intake-stroke
         ?state-name <- (state-name intake)
         (schema current (carburation  ?state3))
         (schema ideal   (carburation  ?state3))
         (schema current (piston-direction ?state4))
         (schema ideal   (piston-direction ?state4))
=>
         (printout t t "1st Stroke - Intake Completed" t t)
         (modify
                (schema current
                        (piston-direction ascending)))
         (modify
                (schema ideal
                        (piston-direction ascending)))
         (retract ?state-name)
         (assert (state-name compression)))

(defrule compression-stroke
         ?state-name <- (state-name compression)
         (schema current (carburation  ?state3))
         (schema ideal   (carburation  ?state3))
         (schema current (piston-direction ?state4))
         (schema ideal   (piston-direction ?state4))
=>
         (printout t t "2nd Stroke - Compression" t t)
         (modify
                (schema current
                        (piston-direction descending)))
         (modify
```

```
                (schema ideal
                        (piston-direction descending)))
        (retract ?state-name)
        (assert (state-name power)))
;       (printout t t "Did sparkplug : fire  or  fail?" t t)
;       (assert (ignition =(read))))

(defrule power-stroke
        ?state-name <- (state-name power)
        ?ignition <- (ignition fire)
        (schema current (carburation  ?state3))
        (schema ideal   (carburation  ?state3))
        (schema current (piston-direction ?state4))
        (schema ideal   (piston-direction ?state4))
=>
        (printout t t "3rd Stroke - Power" t t)
        (modify
                (schema current
                        (piston-direction ascending)))
        (modify
                (schema ideal
                        (piston-direction ascending)))
        (retract ?state-name)
        (assert (state-name exhaust))
        (retract ?ignition))

(defrule exhaust-stroke
        ?state-name <- (state-name exhaust)
        (schema current (carburation  ?state3))
        (schema ideal   (carburation  ?state3))
        (schema current (piston-direction ?state4))
        (schema ideal   (piston-direction ?state4))
=>
        (printout t t "4th Stroke - Exhaust" t t)
        (modify
                (schema current
                        (piston-direction descending)))
        (modify
                (schema ideal
                        (piston-direction descending)))
        (retract ?state-name)
        (assert (state-name intake)))

(defrule switch-plugs
        ?error-flag <- (error-trap ignition)
        ?state-name <- (state-name ?name)
        (schema current (sparkplug2 standby))
=>
        (modify
                (schema current
                        (sparkplug1 fail)
                        (sparkplug2 good)))
;       (retract ?error-flag)
        (retract ?state-name)
        (assert (state-name compression))
        (printout t t "Sparkplug1 set to Failed" t t)
        (printout t t "Sparkplug2 reset from standby to good" t t)
        (printout t t "Is an explanation desired?  yes or no" t t)
        (assert (explanation-flag =(read))))

(defrule no-plugs
        ?error-flag <- (error-trap ignition)
        ?state-name <- (state-name ?name)
        (schema current (sparkplug2 good))
        (schema current (sparkplug1 fail))
=>
        (modify
                (schema current
                        (sparkplug2 fail)))
;       (retract ?error-flag)
        (retract ?name)
        (assert (state-name compression))
        (printout t t "Sparkplug2 set to Failed" t t)
        (printout t t "Is an explanation desired?  yes or no" t t)
        (assert (explanation-flag =(read))))
```

```
(defrule ignition-fire
         (state-name power)
         (split ((sparkplug1 current good)
                   =>)
                  ((sparkplug2 current good)
                   =>))

=>
         (printout t t "Did sparkplug : fire  or  fail?" t t)
         (assert (ignition =(read))))

(defrule explanation-1
         ?error-flag <- (error-trap ignition)
;        ?ignition   <- (ignition ?fire)
         (schema current (sparkplug1 fail))
         (schema current (sparkplug2 good))
         ?expl-flag  <- (explanation-flag yes)
=>
         (printout t t "Sparkplug 1 did not fire. Therefore it was replaced by the backup." t t)
         (retract ?error-flag)
         (retract ?expl-flag))

(defrule explanation-2
         ?error-flag <- (error-trap ignition)
;        ?ignition   <- (ignition ?fire)
         (schema current (sparkplug1 fail))
         (schema current (sparkplug2 fail))
         ?expl-flag  <- (explanation-flag yes)
=>
         (printout t t "Sparkplug 2 did not fire.
Since it is the backup sparkplug,
    (sparkplug 1 has already been considered failed)
       there is no remedy.
However, since it is unusual to have both sparkplugs
   failed, the problem may reside elsewhere." t t)
         (retract ?error-flag)
         (retract ?expl-flag))
```

APPENDIX B

Source Code for System MAIN_SIM_MOD3.ADA

a Communication Amplifier Diagnostic/Explanation Software System

```
    1           package TYP_DEF_SIM is
    2
    3              type SWITCH is (ON,OFF);
    4              type FAULT is (ACTIVE,INACTIVE);
    5              type SOURCE is (A,B);
    6              type CONTINUE is (YES,NO);
    7
    8           end TYP_DEF_SIM;

PSECT MAP

Psect Hex Size     Dec Size     Name
    0 00000037           55    TYP_DEF_SIM_.$CODE
    1 0000000C           12    TYP_DEF_SIM_.$CONSTANT


%ADAC-I-CL_ADDED, Package specification TYP_DEF_SIM added to library
    Replaces older version compiled 17-Nov-1986 00:53


PORTABILITY SUMMARY

      There are no uses of potentially non-portable constructs
```

```
1           -- Main program for Equipment Simulator/Diagnose
2
3           -- Function
4           --   To simulate operation and measurements of a radio frequency
5           --   power source and analize any anomalies. An explanation facility
6           --   is in place.
7           --
8           -- Input
9           --   Initial rf source simulation parameters are read from a file
10          --   by procedure I_O_INIT in package I_O_SIM ; thereafter revised
11          --   values of these parameters are entered interactively via the
12          --   keyboard.
13          --
14          -- Processing
15          --   The input parameter oscillator switch positions are checked for validity
16          --   then, when set properly, the input parameters are fed to the simulator
17          --   package Y_OUT_MOD which at present only consists of procedure EQUIPMENT_Y_O.
18          --   Upon return from this procedure the output power and switch positions
19          --   are monitored. Base upon the readings a tentative diagnosis is made.
20          --   If an explanation is requested, same is supplied via EXPLAIN_DIAGNOSIS.
21          --
22          -- Output
23          --   The results of the diagnosis made by procedures PWR_SWITCH_STATUS and
24          --   OUT_LEVEL in package I_O_SIM_MOD is communicated by them to the CRT
25          --   output device screen.
26
27          -- Algorithm MAIN_SIM_MOD
28          --
29          --   The algorithm used for the driver program is:
30          --
31          --     1. Initialize the simulator by reading the equipment parameters from
32          --        a file.
33          --
34          --     2. While the human monitor desires to continue
35          --        repeat 2 thru 9;
36          --
37          --     3. Test if both A & B sources are switched on and if so force a
38          --        choice;
39          --
40          --     4  Run the simulator in package Y_OUT_MOD;
41          --
42          --     5. Test the results of the simulator (package FAULT_ANALYZE):
44          --       .1  If the output is noise (i.e. -145.0 dbm) then check switches and
45          --          notify the monitor if the master switch is off or if on if both
46          --          the A & B switches are in the off position;
47          --
48          --       .2  If all switches are proper and output is greater than noise but less
49          --           than the output caused by an output amplifier failure then presumably the
50          --           selected oscillator output is below the threshold needed to excite
51          --           the amplifier so advise the monitor to switch to the alternate oscillator;
52          --
53          --       .3  If output is greater than that caused by oscillator substandard
54          --           output but less than that for normal operation then the failure
55          --           of the outputc amplifier may have occurred;
56          --
57          --       .4  If the output is at normal level then report normal operation
```

```
58              --
59              --     6. Write out the equipment status to the CRT (procedure OUT_PARAMS
60              --        in package I_O_SIM_MOD).
61              --
62              --     7. Interactively ask the monitor whether an explanation of the diagnosis
63              --        is desired. If so then supply same.
64              --
65              --     8. Interactively ask the monitor if the simulation is to be continued.
66              --
67              --     9. If the monitor chooses not to continue then exit.
68              --
69              --    10. If the choice is to run another simulation the ask what parameters
70              --        are to be changed (i.e. call NEW_PARAMS in package I_O_SIM_MOD).
71              --
72              --    11. Go back to item 2.
73              --
74              --  end algorithm MAIN_SIM_MOD
75              --
76              --
77              --
78
79
80
81              with TYP_DEF_SIM; with Y_OUT_MOD; with I_O_SIM_MOD; with FAULT_ANALYZE;
82              with EXPLAIN_DIAGNOSIS;
83              use TYP_DEF_SIM; use Y_OUT_MOD;
84
85              Procedure MAIN_SIM_MOD3 is
86
87                AGAIN : CONTINUE:= YES;
88                EXPLAIN : CONTINUE:= YES;
89                EQPT_Y,
90                FREQ_SRC_A,
91                FREQ_SRC_B : SWITCH;
92                REF_FREQ_LEVEL_A,
93                REF_FREQ_LEVEL_B : FLOAT;
94                EQPT_Y_OUT_LEVEL_FAIL : FAULT ;
95                EQUIP_Y_OUT_LEVEL,
96                EQUIP_Y_OUT_LEVEL_SIM,
97                EQUIP_X_OUT_LEVEL : FLOAT;
98
99                begin
100
101               I_O_SIM_MOD.I_O_INIT(EQPT_Y,FREQ_SRC_A,FREQ_SRC_B,EQPT_Y_OUT_LEVEL_FAIL,
102                                    REF_FREQ_LEVEL_A,REF_FREQ_LEVEL_B,
103                                    EQUIP_Y_OUT_LEVEL_SIM,EQUIP_X_OUT_LEVEL);
104
105               while AGAIN = YES
106                  loop
107
108                    if FREQ_SRC_A=ON and FREQ_SRC_B=ON
109                    then I_O_SIM_MOD.I_O_SWITCH(FREQ_SRC_A,FREQ_SRC_B);
110                    end if;  -- If both "ON" force a choice between A or B "ON"
111
112                    I_O_SIM_MOD.NEWPAGE;
113
```

```
114                     EQUIPMENT_Y_O(EQPT_Y,FREQ_SRC_A,FREQ_SRC_B,
115                                   REF_FREQ_LEVEL_A,REF_FREQ_LEVEL_B,EQPT_Y_OUT_LEVEL_FAIL,
116                                   EQUIP_X_OUT_LEVEL,EQUIP_Y_OUT_LEVEL_SIM,
117                                   EQUIP_Y_OUT_LEVEL) ;  -- Run the Simulator
118
119                      FAULT_ANALYZE.DIAGNOSIS(EQPT_Y,FREQ_SRC_A,FREQ_SRC_B,
120                                              EQUIP_Y_OUT_LEVEL_SIM,
121                                              EQUIP_Y_OUT_LEVEL,EQUIP_X_OUT_LEVEL);-- Diagnose fault
122
123                     I_O_SIM_MOD.OUT_PARAMS(EQPT_Y,FREQ_SRC_A,FREQ_SRC_B,EQPT_Y_OUT_LEVEL_FAIL,
124                                            REF_FREQ_LEVEL_A,REF_FREQ_LEVEL_B,
125                                          EQUIP_Y_OUT_LEVEL_SIM,EQUIP_X_OUT_LEVEL,EQUIP_Y_OUT_LEVEL) ;
126
127                     I_O_SIM_MOD.QUERY_EXPLAIN(EXPLAIN); -- Does monitor want an explanation
128
129                     if EXPLAIN =YES
130                     then
131                       EXPLAIN_DIAGNOSIS.EXPLAIN_DIAGNOSIS(EQPT_Y,FREQ_SRC_A,FREQ_SRC_B,REF_FREQ_LEVEL_A,REF_FREQ_LEVEL_B,
132                                                           EQUIP_Y_OUT_LEVEL_SIM,EQUIP_X_OUT_LEVEL,
133                                                    EQUIP_Y_OUT_LEVEL,EXPLAIN) ;  -- If explanation desired
134                     end if;
135
136                     I_O_SIM_MOD.AGAIN_I_O(AGAIN); -- Ask human if wants to repeat simulation
137
138                     exit when AGAIN = NO;
139
140                     I_O_SIM_MOD.NEW_PARAMS(EQPT_Y,FREQ_SRC_A,FREQ_SRC_B,EQPT_Y_OUT_LEVEL_FAIL,
141                                            REF_FREQ_LEVEL_A,REF_FREQ_LEVEL_B,
142                                            EQUIP_Y_OUT_LEVEL_SIM,EQUIP_X_OUT_LEVEL);
143
144                   end loop;
145
146               end MAIN_SIM_MOD3;
```

PSECT MAP

Psect	Hex Size	Dec Size	Name
0	000001F3	499	MAIN_SIM_MOD3.$CODE

%ADAC-I-CL_ADDED, Procedure body MAIN_SIM_MOD3 added to library

PORTABILITY SUMMARY

There are no uses of potentially non-portable constructs

LIBRARY SUMMARY

USER2:[TFL.ADADIR.ADALIB]

Unit name	Nodes	Percent read	Blocks read	Unit kind

```
1   with TYP_DEF_SIM; use TYP_DEF_SIM;
2
3   Package I_O_SIM_MOD is
4
5   -- Procedure to initialize the simulator
6     procedure I_O_INIT(EQPT_Y, FREQ_SRC_A,FREQ_SRC_B : out SWITCH;-- Equipmt sw's
7                        EQPT_Y_OUT_LEVEL_FAIL : out FAULT; -- HW failure
8                        REF_FREQ_LEVEL_A, REF_FREQ_LEVEL_B, -- Osc Output
9                        EQUIP_Y_OUT_LEVEL_SIM, -- HW failure output level
10                       EQUIP_X_OUT_LEVEL : out FLOAT); -- Normal Out
11
12  -- Procedure to test if both or none of ref oscillators are on
13    procedure I_O_SWITCH( FREQ_SRC_A,FREQ_SRC_B : in out SWITCH); -- SW off B or A
14
15  -- Obvious what this does
16    procedure NEWPAGE;
17
18  -- Test and report
19    procedure FAULT_MSG_LIST(MESSAGE_NO : in INTEGER);
20
21  -- Ask if explanation desired
22    procedure QUERY_EXPLAIN(EXPLAIN : out CONTINUE);
23
24
25  -- Present Explanation if requested
26     procedure EXPLAIN_MSG_LIST(MESSAGE_NO :  in INTEGER);
27
28
29  -- Present output parameters all at once
30    procedure OUT_PARAMS(EQPT_Y, FREQ_SRC_A,FREQ_SRC_B : in SWITCH;-- Equipmt sw's
31                         EQPT_Y_OUT_LEVEL_FAIL : in FAULT; -- HW failure
32                         REF_FREQ_LEVEL_A, REF_FREQ_LEVEL_B, -- Osc Output
33                         EQUIP_Y_OUT_LEVEL_SIM, -- HW failure output level
34                         EQUIP_X_OUT_LEVEL, -- Normal Output Power
35                         EQUIP_Y_OUT_LEVEL: in FLOAT); -- Actual Output
36
37
38    procedure PARAM_OUT(PWR_LEVEL : in FLOAT);
39
40
41  -- Read from the keyboard whether eqpt monitor wants to continue the simulation or not
42    procedure AGAIN_I_O(AGAIN : out CONTINUE );
43
44  -- If the simulation is to run again this procedure is called to read in new values
45    procedure NEW_PARAMS(EQPT_Y, FREQ_SRC_A,FREQ_SRC_B : in out SWITCH;-- Equipmt sw's
46                         EQPT_Y_OUT_LEVEL_FAIL : in out FAULT; -- HW failure
47                         REF_FREQ_LEVEL_A, REF_FREQ_LEVEL_B, -- Osc Output
48                         EQUIP_Y_OUT_LEVEL_SIM, -- HW failure output level
49                         EQUIP_X_OUT_LEVEL : in out FLOAT); -- Normal Out
50
51  end I_O_SIM_MOD;
```

```
with TEXT_IO; use TEXT_IO;

Package body I_O_SIM_MOD is

     package SWC_ENUM_IO is new ENUMERATION_IO(SWITCH);
     package SRC_ENUM_IO is new ENUMERATION_IO(SOURCE);
     package FAULT_ENUM_IO is new ENUMERATION_IO(FAULT);
     package AGAIN_IO is new ENUMERATION_IO(CONTINUE);
     package PARAM_IO is new FLOAT_IO(FLOAT);
     INITIAL : FILE_TYPE; -- File of initial parameters for the eqpt simulator

-- This package contains initialization and I/O programs

     procedure I_O_INIT(EQPT_Y, FREQ_SRC_A,FREQ_SRC_B : out SWITCH;-- Equipmt sw's
                    EQPT_Y_OUT_LEVEL_FAIL : out FAULT; -- HW failure
                    REF_FREQ_LEVEL_A, REF_FREQ_LEVEL_B, -- Osc Output
                    EQUIP_Y_OUT_LEVEL_SIM, -- HW failure output level
                    EQUIP_X_OUT_LEVEL : out FLOAT) is  -- Normal Output

               begin  -- Get initial parameter values for the simulator

                  OPEN(INITIAL,IN_FILE,"INITIAL_SIM.DAT");
                  SWC_ENUM_IO.GET(INITIAL,EQPT_Y); SKIP_LINE(INITIAL);
                  SWC_ENUM_IO.GET(INITIAL,FREQ_SRC_A); SKIP_LINE(INITIAL);
                  SWC_ENUM_IO.GET(INITIAL,FREQ_SRC_B); SKIP_LINE(INITIAL);
                  FAULT_ENUM_IO.GET(INITIAL,EQPT_Y_OUT_LEVEL_FAIL); SKIP_LINE(INITIAL);
                  PARAM_IO.GET(INITIAL,REF_FREQ_LEVEL_A); SKIP_LINE(INITIAL);
                  PARAM_IO.GET(INITIAL,REF_FREQ_LEVEL_B); SKIP_LINE(INITIAL);
                  PARAM_IO.GET(INITIAL,EQUIP_Y_OUT_LEVEL_SIM); SKIP_LINE(INITIAL);
                  PARAM_IO.GET(INITIAL,EQUIP_X_OUT_LEVEL);
                  CLOSE(INITIAL);

               end I_O_INIT;

     procedure I_O_SWITCH(FREQ_SRC_A,FREQ_SRC_B : in out SWITCH) is

               FREQ_SRC_SEL : SOURCE;  -- A or B

                      -- This procedure is called from MAIN_SIM
               begin  -- This procedure only called when both A & B switches ar ON

                  PUT(" Both frequency sources A & B are 'ON' & inconsistent");
                  NEW_LINE; PUT( "Which one do you want 'ON'? (type 'A' or 'B')");
                  SRC_ENUM_IO.GET(FREQ_SRC_SEL); SKIP_LINE;NEW_LINE;
                  if FREQ_SRC_SEL = A
                  then FREQ_SRC_B := OFF;
                  else FREQ_SRC_A := OFF;
                  end if;

               end I_O_SWITCH; -- This procedure shouldn't have logic but pass
                               -- FREQ_SRC_SEL to another package supporting the simulation. May change this later


      procedure NEWPAGE is
            begin
              NEW_PAGE;
            end NEWPAGE;
```

```
procedure FAULT_MSG_LIST(MESSAGE_NO : in INTEGER) is
        -- Test all on/off switch stati, then decide upon
     begin -- what diagnosis to transmit to equipment monitor

     case MESSAGE_NO is

         when 1 =>
          NEW_LINE;PUT(" Equipment output power is down in the noise level."); -- i.e -125 dbm
           NEW_LINE;

         when 2  =>
          NEW_LINE;PUT(" The main switch is 'ON' so an internal failure occurred" );
           NEW_LINE;

         when 3  =>
          NEW_LINE;PUT("Both oscillators A and B are OFF so no wonder !!");
           NEW_LINE;PUT( " Next pass try turning on either oscillator.");
           NEW_LINE;

         when 4  =>
          NEW_LINE;PUT("The selected oscillator may have completely failed");
           NEW_LINE;PUT(" Next pass try turning on the other oscillator");
           NEW_LINE;

         when 5  =>
          NEW_LINE;PUT( "The Main switch is 'OFF' so no wonder the output is noise");
           NEW_LINE;

         when 6  =>
           NEW_LINE;PUT("The OPERATIONAL DIAGNOSIS is");
           NEW_LINE;PUT("Since the power output is ");
           NEW_LINE;

         when 7  =>
           NEW_LINE; PUT(" The selected oscillator level is probably below threshold ");
           NEW_LINE; PUT(" so next pass switch to the other oscillator.");
           NEW_LINE;

         when 8  =>
           NEW_LINE; PUT(" A degradation of output has occurred ");
           NEW_LINE; PUT(" probably NOT caused by an oscillator failure ");
           NEW_LINE; PUT(" switch to backup unit if it's available");
           NEW_LINE;

         when 9  =>
           NEW_LINE; PUT("Normal operation with normal output. ");
           NEW_LINE;

         when others =>
           NEW_LINE; PUT(" ERROR in value of MESSAGE_NO");NEW_LINE;

       end case;

     end FAULT_MSG_LIST;
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
115
116
117             procedure QUERY_EXPLAIN(EXPLAIN : out CONTINUE) is
118
119                  begin -- This communicates with the human monitor asking if an
120                        -- explanation desired
121                    PUT(" Do you want to know the reasoning behind the diagnostic decision? (YES/NO)");
122                    AGAIN_IO.GET(EXPLAIN);NEW_LINE(2);
123
124                  end QUERY_EXPLAIN;
125
126
127             procedure EXPLAIN_MSG_LIST(MESSAGE_NO : in INTEGER) is
128
129                  begin -- This procedure contains some of the diagnostic logic
130                        -- for analizing a substandard power output
131
132                    case MESSAGE_NO is
133
134                      when 10 =>
135
136                           NEW_LINE;PUT(" The OPERATIONAL DIAGNOSIS EXPLANATION is:");
137                           NEW_LINE;
138
139                      when 11  =>
140
141                           NEW_LINE;PUT(" The output measured was very low and noisy");
142                           NEW_LINE;PUT(" yet the master switch was 'ON',");
143                           NEW_LINE;PUT(" so the failure was something else.");
144
145                      when 12  =>
146
147                           NEW_LINE;PUT( " We then found that both the oscillator switches were 'OFF',");
148                           NEW_LINE;PUT( "so the malfunction was a set-up error.");
149                           NEW_LINE;
150
151                      when 13  =>
152
153                           NEW_LINE;PUT(" We found the selected oscillator switch to be 'ON',");
154                           NEW_LINE;PUT(" so,since output is noise,potentially we had a catastrophic oscillator failure");
155                           NEW_LINE;
156
157                      when 14  =>
158
159                           NEW_LINE;PUT(" We found the master switch in the 'OFF' position so ");
160                           NEW_LINE;PUT(" the malfunction was a set-up error. ");
161                           NEW_LINE;
162
163                      when 15  =>
164
165                           NEW_LINE;PUT(" The output was between -145.0 dbm and -90.0 dbm. Thus the output");
166                           NEW_LINE;PUT(" device was probably OK but the oscillator was below threshold.");
167                           NEW_LINE;PUT(" The oscillator output was insufficient to drive the output device.");
168                           NEW_LINE;PUT(" The logical action is to select the other oscillator on the next pass.");
169                           NEW_LINE;
170
```

```
171                     when 16  =>
173                          NEW_LINE;PUT(" A degradation of output had occurred; however the output was above-90.0 dbm");
174                          NEW_LINE;PUT(" but this is the output when output device failure occurs.");
175                          NEW_LINE;PUT(" This indicates that the selected oscillator was probably OK");
176                          NEW_LINE;PUT(" since there was some signal leaking through the circuitry,");
177                          NEW_LINE;PUT(" but there was probably an output device failure.");
178                          NEW_LINE;
179
180                     when 17  =>
181
182                          NEW_LINE;PUT(" Normal operation with normal output has occurred.");
183                          NEW_LINE;
184
185                      when others =>
186                          NEW_LINE; PUT(" ERROR in EXPLAIN_DIAGNOSIS with MESSAGE_NO.");
187                          NEW_LINE;
188
189                    end case;
190
191                end EXPLAIN_MSG_LIST;
192
193
194             procedure OUT_PARAMS(EQPT_Y, FREQ_SRC_A,FREQ_SRC_B : in SWITCH;-- Equipmt sw's
195                                  EQPT_Y_OUT_LEVEL_FAIL : in FAULT; -- HW failure
196                                  REF_FREQ_LEVEL_A, REF_FREQ_LEVEL_B, -- Osc Output
197                                  EQUIP_Y_OUT_LEVEL_SIM, -- HW failure output level
198                                  EQUIP_X_OUT_LEVEL ,  -- Normal Output Power
199                                  EQUIP_Y_OUT_LEVEL : in FLOAT) is
200
201                        begin -- This procedure outputs the status of all simulation variable
202                              -- database for the current pass of the simulator code
203
204                          NEW_LINE;
205                          PUT("The Status of the system is :");NEW_LINE(2);
206                          PUT("Equipment Switch is ");SWC_ENUM_IO.PUT(EQPT_Y);NEW_LINE;
207                          PUT("Frequency Source A is ");SWC_ENUM_IO.PUT(FREQ_SRC_A); -- Switches
208                          PUT(";  Frequency Source B is ");SWC_ENUM_IO.PUT(FREQ_SRC_B);NEW_LINE;
209                          PUT("The actual power output of the system is ");
210                          PARAM_IO.PUT(EQUIP_Y_OUT_LEVEL);NEW_LINE;
211                          NEW_LINE(2);
212                          PUT(" The status of the equipment and simulator data is: ");
213                          NEW_LINE;PUT(" (these data normally unavailable to observer)");
214                          NEW_LINE(2);
215                          PUT("The GREMLIN is ");FAULT_ENUM_IO.PUT(EQPT_Y_OUT_LEVEL_FAIL);NEW_LINE;
216                          PUT("The GREMLIN, if active causes system output of <= "); -- power lpower levels
217                          PARAM_IO.PUT(EQUIP_Y_OUT_LEVEL_SIM);
218                          NEW_LINE;
219                          PUT("The Oscillator A normal output is ");PARAM_IO.PUT(REF_FREQ_LEVEL_A);
220                          NEW_LINE;
221                          PUT("The Oscillator B normal output is ");PARAM_IO.PUT(REF_FREQ_LEVEL_B);
222                          NEW_LINE;
223                          PUT("The normal system output is ");PARAM_IO.PUT(EQUIP_X_OUT_LEVEL);
224                          NEW_LINE(2);
225
226                       end OUT_PARAMS;
```

```
228             procedure PARAM_OUT(PWR_LEVEL : in FLOAT) is
230                           begin
231                              PARAM_IO.PUT(PWR_LEVEL);
232                           end PARAM_OUT;
233
234
235
236             procedure AGAIN_I_O(AGAIN: out CONTINUE) is
237
238                       begin       -- Ask the equipment monitor whether or not to continue
239                         NEW_LINE;
240                         PUT(" Do you wish to change some parameters and try again ? ");
241                         PUT( "(type in YES/NO) ");AGAIN_IO.GET(AGAIN); SKIP_LINE;
242                         NEW_LINE(2);
243                       end AGAIN_I_O;
244
245             procedure NEW_PARAMS(EQPT_Y, FREQ_SRC_A,FREQ_SRC_B : in out SWITCH;-- Equipmt sw's
246                                  EQPT_Y_OUT_LEVEL_FAIL : in out FAULT; -- HW failure
247                                  REF_FREQ_LEVEL_A, REF_FREQ_LEVEL_B, -- Osc output
248                                  EQUIP_Y_OUT_LEVEL_SIM, -- HW failure output level
249                                  EQUIP_X_OUT_LEVEL : in out FLOAT) is  -- Normal out
250
251                        SubType INPUT_STRING is STRING(1..9);
252                        STRLEN : NATURAL ; -- Length of the string read from the keyboard buffer
253                        LOCAL_STRING: INPUT_STRING; -- Declaration of max length to be read
254                        CONVLEN : POSITIVE ; -- Dummy local var containing len of enum attrib
255
256                        begin -- Display present parameter values and solicit new ones
257
258                          PUT("Enter new values and <CR> or just <CR> to let value stand as is");NEW_LINE(2);
259
260             -- Show master switch position (ON/OFF) and ask for new value
261
262                          PUT("Equipment switch is  ");SWC_ENUM_IO.PUT(EQPT_Y);NEW_LINE;
263                          PUT("Enter new value (ON/OFF) ");
264                          GET_LINE(LOCAL_STRING,STRLEN); -- Get new input
265                          if STRLEN /= INPUT_STRING'FIRST-1          -- if an input (i.e. not  just a <CR>)
266                          then SWC_ENUM_IO.GET(LOCAL_STRING,EQPT_Y,CONVLEN); -- then enter into DBMS
267                          end if; LOCAL_STRING := "         ";NEW_LINE; -- reinitialize string object
268
269             -- Show Frequency sources switch positions and ask for new values
270
271                          PUT("Freq source A is  ");SWC_ENUM_IO.PUT(FREQ_SRC_A);NEW_LINE;
272                          PUT("Enter new value (ON/OFF) ");GET_LINE(LOCAL_STRING,STRLEN);
273                          if STRLEN /= INPUT_STRING'FIRST-1                -- if an input then get it as a string
274                          then SWC_ENUM_IO.GET(LOCAL_STRING,FREQ_SRC_A,CONVLEN); -- convert string to its enumeration type
275                          end if; LOCAL_STRING := "         "; NEW_LINE;
276                          PUT("Freq source B is  ");SWC_ENUM_IO.PUT(FREQ_SRC_B);NEW_LINE;
277                          PUT("Enter new value (ON/OFF) ");GET_LINE(LOCAL_STRING,STRLEN);
278                          if STRLEN /= INPUT_STRING'FIRST-1
279                          then SWC_ENUM_IO.GET(LOCAL_STRING,FREQ_SRC_B,CONVLEN);
280                          end if; LOCAL_STRING := "         ";NEW_LINE;
281
282             -- Present the simulated failure status (ACTIVE/INACTIVE) ask which?
283
284                          PUT("The GREMLIN is  ");FAULT_ENUM_IO.PUT(EQPT_Y_OUT_LEVEL_FAIL);
285                          NEW_LINE;
```

```
286                     PUT(" Enter new value (ACTIVE/INACTIVE) ");GET_LINE(LOCAL_STRING,STRLEN);
287                     if STRLEN /= INPUT_STRING'FIRST-1
288                     then FAULT_ENUM_IO.GET(LOCAL_STRING,EQPT_Y_OUT_LEVEL_FAIL,CONVLEN);
289                     end if; LOCAL_STRING := "          ";NEW_LINE;
290
291             -- Present the output power levels for each oscillator and ask for updates
292
293                     PUT("The Oscillator A output is ");PARAM_IO.PUT(REF_FREQ_LEVEL_A);
294                     NEW_LINE;
295                     PUT(" Enter a new value for Oscillator A power output in dbm ");NEW_LINE;
296                     PUT(" (must be > -145.0 ) ");  GET_LINE(LOCAL_STRING,STRLEN);
297                     if STRLEN /= INPUT_STRING'FIRST-1
298                     then PARAM_IO.GET(LOCAL_STRING,REF_FREQ_LEVEL_A,CONVLEN);
299                     end if; LOCAL_STRING := "          "; NEW_LINE;
300                     PUT("The Oscillator B output is ");PARAM_IO.PUT(REF_FREQ_LEVEL_B);
301                     NEW_LINE;
302                     PUT(" Enter a new value for Oscillator B power output in dbm ");NEW_LINE;
303                     PUT(" ( must be > -145.0  "); GET_LINE(LOCAL_STRING,STRLEN);
304                     if STRLEN /= INPUT_STRING'FIRST-1
305                     then PARAM_IO.GET(LOCAL_STRING,REF_FREQ_LEVEL_B,CONVLEN);
306                     end if; LOCAL_STRING := "          "; NEW_LINE;
307
308             -- Present the normal operation power and ask for parameter change
309
310                     PUT("The normal system power output is ");PARAM_IO.PUT(EQUIP_X_OUT_LEVEL);
311                     NEW_LINE;
312                     PUT(" Enter a new value for system power output if desired, ");
313                     GET_LINE(LOCAL_STRING,STRLEN);
314                     if STRLEN /= INPUT_STRING'FIRST-1
315                     then PARAM_IO.GET(LOCAL_STRING,EQUIP_X_OUT_LEVEL,CONVLEN);
316                     end if; LOCAL_STRING := "          ";NEW_LINE;
317
318             -- Present the ouput power level when an output device fails (dbm) and inquire
319
320                     PUT("The output power when output device has failed,");NEW_LINE;
321                     PUT(" i.e. ACTIVE GREMLIN is  "); PARAM_IO.PUT(EQUIP_Y_OUT_LEVEL_SIM);
322                     NEW_LINE;
323                     PUT(" Enter a new value for this 'GREMLIN' power out if desired,");
324                     NEW_LINE;PUT("note that it must be less than normal power out,");
325                     NEW_LINE;PUT(" but must also be > -90.0 ");GET_LINE(LOCAL_STRING,STRLEN);
326                     if STRLEN /= INPUT_STRING'FIRST-1
327                     then PARAM_IO.GET(LOCAL_STRING,EQUIP_Y_OUT_LEVEL_SIM,CONVLEN);
328                     end if; LOCAL_STRING := "          ";NEW_LINE(2);
329
330             -- Ready to loop back and do it again
331
332                     PUT_LINE("  Ready or not; Here we go again !! ");
333                     NEW_LINE;
334
335                 end NEW_PARAMS;
336             end I_O_SIM_MOD;
```

```
     1          with TYP_DEF_SIM;use TYP_DEF_SIM;
     2          Package Y_OUT_MOD is
     3          procedure EQUIPMENT_Y_O(EQPT_Y : in SWITCH; -- Equipment Switch
     4                                  FREQ_SRC_A,FREQ_SRC_B: in SWITCH; -- Osc Sw's
     5                                  REF_FREQ_LEVEL_A,
     6                                  REF_FREQ_LEVEL_B :  in FLOAT; -- Osc Outputs in dbm
     7                                  EQPT_Y_OUT_LEVEL_FAIL : in FAULT; -- Fail Flag
     8                                  EQUIP_X_OUT_LEVEL, -- Normal output > 20.0
     9                                  EQUIP_Y_OUT_LEVEL_SIM : in FLOAT; -- Fail Out Pwr < 20.0
    10                                  EQUIP_Y_OUT_LEVEL : out FLOAT); -- Equip Pwr Out (actual)
    11
    12          end Y_OUT_MOD;
    13

Line 2
                                        0000    Y_OUT_MOD_$ELAB:
                                  C000  0000            .entry      Y_OUT_MOD_$ELAB,^m<dv,iv>
                              5E 0C C2  0002            subl2       #12,sp
                          5C 00000000 EF  DE    0005    moval       $CONSTANT,ap
                              5C 6C DE      000C        moval       (ap),ap
                              5C 5C D0      000F        movl        ap,ap
                           F4 AD 5C D0      0012        movl        ap,-12(fp)
                       5C 00000000* EF 7E   0016        movaq       ADA$HANDLER,ap
                              6D 5C D0      001D        movl        ap,(fp)
                          5C 00000000 EF DE 0020        moval       $CONSTANT,ap
                              5C 04 AC 9E   0027        movab       4(ap),ap
                              5C 5C D0      002B        movl        ap,ap
                           F4 AD 5C D0      002E        movl        ap,-12(fp)

Line 13
                           F4 AD 00 D0      0032        movl        #0,-12(fp)
                                            0036    Y_OUT_MOD_$ELAB$RTN_LBL25:
                                     04     0036        ret

PSECT MAP

Psect Hex Size     Dec Size     Name
    0 00000037           55     Y_OUT_MOD_.$CODE
    1 0000000C           12     Y_OUT_MOD_.$CONSTANT


%ADAC-I-CL_ADDED, Package specification Y_OUT_MOD added to library
    Replaces older version compiled 27-Nov-1986 00:15


PORTABILITY SUMMARY

      There are no uses of potentially non-portable constructs




LIBRARY SUMMARY

USER2:[TFL.ADADIR.ADALIB]
```

```
     1          Package body Y_OUT_MOD is
     2          procedure EQUIPMENT_Y_O(EQPT_Y : in SWITCH; -- Equipment Switch
     3                                  FREQ_SRC_A,FREQ_SRC_B: in SWITCH; -- Osc Sw's
     4                                  REF_FREQ_LEVEL_A,
     5                                  REF_FREQ_LEVEL_B :  in FLOAT; -- Osc Output in dbm
     6                                  EQPT_Y_OUT_LEVEL_FAIL : in FAULT; -- Fail Flag
     7                                  EQUIP_X_OUT_LEVEL, -- Normal output > 20.0
     8                                  EQUIP_Y_OUT_LEVEL_SIM : in FLOAT; -- Fail Out Pwr
     9                                  EQUIP_Y_OUT_LEVEL : out FLOAT) is -- Equip Pwr Out (actual)
    10
    11          Ref_freq_level, Equip_Y_output_internal: FLOAT ; -- Internal variables
    12
    13          begin
    14             if EQPT_Y = ON
    15             then if FREQ_SRC_A = ON
    16                  then Ref_freq_level := REF_FREQ_LEVEL_A;
    17                  else if FREQ_SRC_B = ON
    18                       then Ref_freq_level := REF_FREQ_LEVEL_B;
    19                       else Ref_freq_level := -145.0;
    20                       end if;
    21                  end if;
    22                 if (FREQ_SRC_A = ON) and (REF_FREQ_LEVEL_A <= -120.0)
    23                  then Ref_freq_level := -145.0;
    24                  else if (FREQ_SRC_B = ON) and (REF_FREQ_LEVEL_B <= -120.0)
    25                       then Ref_freq_level := -145.0;
    26                       end if;
    27                  end if;
    28                  if Ref_freq_level <= -145.0
    29                  then Equip_Y_output_internal := -145.0;
    30                  else if Ref_freq_level < -5.0
    31                       then Equip_Y_output_internal := -90.0;
    32                       else if EQPT_Y_OUT_LEVEL_FAIL = ACTIVE
    33                            then Equip_Y_output_internal := EQUIP_Y_OUT_LEVEL_SIM;
    34                            else Equip_Y_output_internal := EQUIP_X_OUT_LEVEL;
    35                            end if;
    36                       end if;
    37                  end if;
    38                  EQUIP_Y_OUT_LEVEL := Equip_Y_output_internal;
    39              else EQUIP_Y_OUT_LEVEL := -145.0;
    40              end if;
    41          end EQUIPMENT_Y_O;
    42          end Y_OUT_MOD;

PSECT MAP

Psect Hex Size     Dec Size     Name
    0 0000010B          267     Y_OUT_MOD.$CODE
    1 0000000C           12     Y_OUT_MOD.$CONSTANT
    2 00000001            1     Y_OUT_MOD.$DATA


%ADAC-I-CL_ADDED, Package body Y_OUT_MOD added to library
    Corresponds to package specification Y_OUT_MOD compiled 27-Nov-1986 00:25
```

```
    1
    2             with TYP_DEF_SIM; use TYP_DEF_SIM;
    3             with I_O_SIM_MOD;
    4            package FAULT_ANALYZE is
    5
    6              procedure SWITCH_STATUS(EQPT_Y,FREQ_SRC_A,FREQ_SRC_B : in SWITCH) ;
    7
    8              procedure OUT_LEVEL(EQUIP_Y_OUT_LEVEL_SIM,
    9                                  EQUIP_Y_OUT_LEVEL,
   10                                  EQUIP_X_OUT_LEVEL : in FLOAT) ;
   11
   12              procedure DIAGNOSIS(EQPT_Y,FREQ_SRC_A,FREQ_SRC_B : in SWITCH;
   13                                  EQUIP_Y_OUT_LEVEL_SIM,
   14                                  EQUIP_Y_OUT_LEVEL,
   15                                  EQUIP_X_OUT_LEVEL : in FLOAT) ;
   16
   17
   18            end FAULT_ANALYZE;

PSECT MAP

Psect Hex Size    Dec Size    Name
    0 00000020          32    FAULT_ANALYZE_.$CODE
    1 0000000C          12    FAULT_ANALYZE_.$CONSTANT


%ADAC-I-CL_ADDED, Package specification FAULT_ANALYZE added to library
    Replaces older version compiled 18-Jan-1987 21:21


PORTABILITY SUMMARY

      There are no uses of potentially non-portable constructs




LIBRARY SUMMARY

USER2:[TFL.ADADIR.ADALIB]

      Unit name              Nodes      Percent      Blocks                      Unit kind
                                          read                  read
TYP_DEF_SIM                      7     41              7              Package specification
I_O_SIM_MOD                      4      7              5              Package specification
```

```
19
20
21          package body FAULT_ANALYZE is
22
23            MESSAGE_NO : INTEGER;
24
25            procedure SWITCH_STATUS(EQPT_Y,FREQ_SRC_A,FREQ_SRC_B : in SWITCH) is
26
27                        -- Called by MAIN_SIM
28                        -- Test all on/off switch stati, then decide upon
29                    begin -- what diagnosis to transmit to equipment monitor
30
31                       MESSAGE_NO := 1 ; -- PUT(" Equipment output power is down in the noise level.")
32                       I_O_SIM_MOD.FAULT_MSG_LIST(MESSAGE_NO);
33                       if EQPT_Y = ON
34                       then
35                         MESSAGE_NO := 2  ; -- Main switch is on so must be other failure
36                         I_O_SIM_MOD.FAULT_MSG_LIST(MESSAGE_NO);
37                         if (FREQ_SRC_A=OFF) and (FREQ_SRC_B=OFF)
38                         then
39                           MESSAGE_NO := 3;
40                                --  PUT("Both oscillators A and B are OFF so no wonder !!");
41                                --  PUT( " Next pass try turning on either oscillator.");
42                           I_O_SIM_MOD.FAULT_MSG_LIST(MESSAGE_NO);
43                         else
44                           MESSAGE_NO := 4; -- Selected oscillator must Bbe inoperative so
45                                            -- Next time select the other oscillator
46                           I_O_SIM_MOD.FAULT_MSG_LIST(MESSAGE_NO);
47                         end if;
48                       else
49                           MESSAGE_NO := 5; -- Main Switch is 'OFF' that explains the poor output
50                           I_O_SIM_MOD.FAULT_MSG_LIST(MESSAGE_NO);
51                       end if;
52                    end;
53
54             procedure OUT_LEVEL(EQUIP_Y_OUT_LEVEL_SIM, EQUIP_Y_OUT_LEVEL,
55                                 EQUIP_X_OUT_LEVEL : in FLOAT) is
56
57                    begin -- This procedure contains some of the diagnostic logic
58                          -- for analizing a substandard power output
59                       MESSAGE_NO := 6;
60                                    -- PUT(" The OPERATIONAL DIAGNOSIS is:");
61                                    -- PUT("The power output is  ");
62                       I_O_SIM_MOD.FAULT_MSG_LIST(MESSAGE_NO);
63                       I_O_SIM_MOD.PARAM_OUT(EQUIP_Y_OUT_LEVEL);
64                       if EQUIP_Y_OUT_LEVEL <= -90.0
65                       then
66                         MESSAGE_NO := 7;
67                           -- PUT(" The reference oscillator level is below threshold ")
68                           -- PUT(" so next pass choose the other oscillator. ");
69                         I_O_SIM_MOD.FAULT_MSG_LIST(MESSAGE_NO);
70                       else
71                         if EQUIP_Y_OUT_LEVEL  <= EQUIP_Y_OUT_LEVEL_SIM
72                         then
73                           MESSAGE_NO := 8;
74                                -- PUT("A degradation of output has occurred,");
75                                -- PUT("caused by an output device Failure.");
```

```
                            I_O_SIM_MOD.FAULT_MSG_LIST(MESSAGE_NO);
                          else
                            if EQUIP_Y_OUT_LEVEL <= EQUIP_X_OUT_LEVEL
                            then
                              MESSAGE_NO := 9;
                                --PUT(" Normal operation with normal output ");
                              I_O_SIM_MOD.FAULT_MSG_LIST(MESSAGE_NO);
                            end if;
                          end if;
                        end if;
                      end OUT_LEVEL;


              procedure DIAGNOSIS(EQPT_Y,FREQ_SRC_A,FREQ_SRC_B : in SWITCH;
                                  EQUIP_Y_OUT_LEVEL_SIM,
                                  EQUIP_Y_OUT_LEVEL,
                                  EQUIP_X_OUT_LEVEL : in FLOAT) is
                 begin

                      if EQUIP_Y_OUT_LEVEL <= -145.0

                      then SWITCH_STATUS(EQPT_Y,FREQ_SRC_A,FREQ_SRC_B);

                      else OUT_LEVEL(EQUIP_Y_OUT_LEVEL_SIM, EQUIP_Y_OUT_LEVEL,
                                     EQUIP_X_OUT_LEVEL) ;

                      end if;

                  end DIAGNOSIS;

            end FAULT_ANALYZE ;

PSECT MAP

Psect Hex Size     Dec Size     Name
    0 00000269          617     FAULT_ANALYZE.$CODE
    1 0000000C           12     FAULT_ANALYZE.$CONSTANT
    2 00000008            8     FAULT_ANALYZE.$DATA


%ADAC-I-CL_ADDED, Package body FAULT_ANALYZE added to library
    Corresponds to package specification FAULT_ANALYZE compiled 18-Jan-1987 21:22


PORTABILITY SUMMARY

      There are no uses of potentially non-portable constructs




LIBRARY SUMMARY

USER2:[TFL.ADADIR.ADALIB]
```

```
     1          with TYP_DEF_SIM; use TYP_DEF_SIM;
     2          with I_O_SIM_MOD;
     3
     4          Package EXPLAIN_DIAGNOSIS is
     5
     6               procedure EXPLAIN_DIAGNOSIS(EQPT_Y, FREQ_SRC_A,FREQ_SRC_B : in SWITCH;-- Equipmt sw's
     7                                           REF_FREQ_LEVEL_A, REF_FREQ_LEVEL_B, -- Osc Out
     8                                           EQUIP_Y_OUT_LEVEL_SIM, -- HW failure out level
     9                                           EQUIP_X_OUT_LEVEL ,  -- Normal Out Power
    10                                           EQUIP_Y_OUT_LEVEL : in FLOAT;
    11                                           EXPLAIN : in out CONTINUE) ;
    12
    13          end EXPLAIN_DIAGNOSIS;

PSECT MAP

Psect Hex Size     Dec Size     Name
    0 00000020           32     EXPLAIN_DIAGNOSIS_.$CODE
    1 0000000C           12     EXPLAIN_DIAGNOSIS_.$CONSTANT


%ADAC-I-CL_ADDED, Package specification EXPLAIN_DIAGNOSIS added to library
    Replaces older version compiled 11-Jan-1987 02:22


PORTABILITY SUMMARY

      There are no uses of potentially non-portable constructs




LIBRARY SUMMARY

USER2:[TFL.ADADIR.ADALIB]

      Unit name                Nodes      Percent      Blocks                      Unit kind
                                            read                 read
TYP_DEF_SIM                       10     58             7              Package specification
I_O_SIM_MOD                        4      7             5              Package specification
```

```
14
15          Package body Explain_Diagnosis is
16
17               procedure EXPLAIN_DIAGNOSIS(EQPT_Y, FREQ_SRC_A,FREQ_SRC_B : in SWITCH;-- Equipmt sw's
18                                 REF_FREQ_LEVEL_A, REF_FREQ_LEVEL_B, -- Osc Out
19                                 EQUIP_Y_OUT_LEVEL_SIM, -- HW failure out level
20                                 EQUIP_X_OUT_LEVEL ,  -- Normal Out Power
21                                 EQUIP_Y_OUT_LEVEL : in FLOAT;
22                                 EXPLAIN : in out CONTINUE) is
23
24                    MESSAGE_NO : INTEGER;  -- Local variable to contain number of msg
25                                           -- for the message print procedure
26
27               -- This procedure provides an explanation of the reasoning behind the
28               -- diagnosis of the equipment's ills.
29               -- It is called from MAIN_SIM.
30
31                    begin
32
33                      MESSAGE_NO := 10;
34                      I_O_SIM_MOD.EXPLAIN_MSG_LIST(MESSAGE_NO);-- Print Header
35                           if EQUIP_Y_OUT_LEVEL <= -145.0
36                           then
37                              if EQPT_Y = ON
38                              then
39                                MESSAGE_NO := 11;
40                                 -- The output measured was very low and noisy,);
41                                 --  yet the master switch was ON.);
42                                 --  Obviously was some other failure.);
43                                I_O_SIM_MOD.EXPLAIN_MSG_LIST(MESSAGE_NO);
44                                if (FREQ_SRC_A=OFF) and (FREQ_SRC_B=OFF)
45                                then
46                                  MESSAGE_NO := 12;
47                                      -- We then found both the oscillator switches were OFF, );
48                                      -- so the malfunction was a set_up error.);
49                                  I_O_SIM_MOD.EXPLAIN_MSG_LIST(MESSAGE_NO);
50                                else
51                                  MESSAGE_NO := 13;
52                                      -- (We found the selected oscillator switch to be ON,);
53                                      -- (so potentially we had a catastrophic oscillator failure.);
54                                  I_O_SIM_MOD.EXPLAIN_MSG_LIST(MESSAGE_NO);
55                                end if;
56                             else
57                                 MESSAGE_NO := 14;
58                                     -- (The master switch was found to be OFF so the malfunction);
59                                     -- (was a set_up error.)
60                                 I_O_SIM_MOD.EXPLAIN_MSG_LIST(MESSAGE_NO);
61                             end if;
62                          else
63                            if EQUIP_Y_OUT_LEVEL <= -90.0
64                            then
65                              MESSAGE_NO := 15;
66                                  -- (The out was between -145.0 dbm and -90.0 dbm. Thus the out );
67                                  -- ( device was probably OK but the oscillator was below threshold.);
68                                  -- (The oscillator out was insufficient to drive the out device);
69                                  -- (Obviously on the next pass one should choose the other oscillator. )
70                               I_O_SIM_MOD.EXPLAIN_MSG_LIST(MESSAGE_NO);
```

```
                        else
                          if EQUIP_Y_OUT_LEVEL  <= EQUIP_Y_OUT_LEVEL_SIM
                          then
                            MESSAGE_NO := 16;
                              -- (A degradation of out had occurred, the out was above);
                              --( -90.0 dbm but was still less than or equal to the out failure dbm.);
                              --(This indicates that the selected oscillator was OK,);
                              --( since some signal was 'leaking' through the circuitry;);
                              --( but there was probably an out device Failure.);
                            I_O_SIM_MOD.EXPLAIN_MSG_LIST(MESSAGE_NO);
                          else
                            if EQUIP_Y_OUT_LEVEL <= EQUIP_X_OUT_LEVEL
                            then
                              MESSAGE_NO := 17;
                                --( Normal operation with normal or near normal out );
                                --( since the out is above the out failure level.)
                              I_O_SIM_MOD.EXPLAIN_MSG_LIST(MESSAGE_NO);
                            end if;
                          end if;
                        end if;
                      end if;
                end EXPLAIN_DIAGNOSIS;
            end Explain_Diagnosis;
```

PSECT MAP

Psect	Hex Size	Dec Size	Name
0	000000F9	249	Explain_Diagnosis.$CODE
1	0000000C	12	Explain_Diagnosis.$CONSTANT
2	00000001	1	Explain_Diagnosis.$DATA

%ADAC-I-CL_ADDED, Package body EXPLAIN_DIAGNOSIS added to library
    Replaces older version compiled 11-Jan-1987 02:22
    Corresponds to package specification EXPLAIN_DIAGNOSIS compiled 11-Jan-1987 02:29

PORTABILITY SUMMARY

There are no uses of potentially non-portable constructs

LIBRARY SUMMARY

USER2:[TFL.ADADIR.ADALIB]

Unit name	Nodes	Percent read	Blocks read	Unit kind
EXPLAIN_DIAGNOSIS	16	100		Package specification
TYP_DEF_SIM	16	94	9	Package specification
I_O_SIM_MOD	7	12	8	Package specification

N
N
OFF
CTIVE
3.0
3.0
-60.0
0.0

Object Module Synopsis

Module Name	Ident	Bytes	File	Creation Date	Creator
ADA$ELAB_MAIN_SIM_MOD3	01	82	[TFL.ADADIR]MAIN_SIM_MOD3.OBJ;1	20-Jan-1987 00:52	VAX Ada V1.3-23
TYP_DEF_SIM_	01	67	TYP_DEF_SIM_.OBJ;2	17-NOV-1986 15:34	VAX Ada V1.3-23
IO_EXCEPTIONS_	01	6	[SYSLIB.ADALIB]IO_EXCEPTIONS_.OBJ;1	6-MAR-1985 12:40	VAX Ada V1.0-7
Y_OUT_MOD_	01	67	[TFL.ADADIR.ADALIB]Y_OUT_MOD_.OBJ;2	27-NOV-1986 00:25	VAX Ada V1.3-23
Y_OUT_MOD	01	280	[TFL.ADADIR.ADALIB]Y_OUT_MOD.OBJ;1	18-JAN-1987 22:18	VAX Ada V1.3-23
TEXT_IO_	01	31	[SYSLIB.ADALIB]TEXT_IO_.OBJ;1	6-MAR-1985 12:42	VAX Ada V1.0-7
TEXT_IO	01	184	[SYSLIB.ADALIB]TEXT_IO.OBJ;1	6-MAR-1985 12:43	VAX Ada V1.0-7
I_O_SIM_MOD_	01	44	I_O_SIM_MOD_.OBJ;5	11-JAN-1987 00:13	VAX Ada V1.3-23
I_O_SIM_MOD	01	14608	I_O_SIM_MOD.OBJ;14	20-JAN-1987 00:51	VAX Ada V1.3-23
EXPLAIN_DIAGNOSIS_	01	44	EXPLAIN_DIAGNOSIS_.OBJ;5	11-JAN-1987 02:29	VAX Ada V1.3-23
EXPLAIN_DIAGNOSIS	01	262	EXPLAIN_DIAGNOSIS.OBJ;3	11-JAN-1987 02:29	VAX Ada V1.3-23
FAULT_ANALYZE_	01	44	FAULT_ANALYZE_.OBJ;3	18-JAN-1987 21:22	VAX Ada V1.3-23
FAULT_ANALYZE	01	637	FAULT_ANALYZE.OBJ;1	18-JAN-1987 21:22	VAX Ada V1.3-23
MAIN_SIM_MOD3	01	499	MAIN_SIM_MOD3.OBJ;1	18-JAN-1987 22:31	VAX Ada V1.3-23

Program Section Synopsis

Psect Name	Module Name	Base	End	Length		Align	Attributes
$CONSTANT		00000200	00001147	00000F48	(3912.)	LONG 2	PIC,USR,CON,REL,LCL, SHR,NOEXE, RD,NOWRT,NOVEC
	TYP_DEF_SIM_	00000200	0000020B	0000000C	(12.)	LONG 2	
	Y_OUT_MOD_	0000020C	00000217	0000000C	(12.)	LONG 2	
	Y_OUT_MOD	00000218	00000223	0000000C	(12.)	LONG 2	
	I_O_SIM_MOD_	00000224	0000022F	0000000C	(12.)	LONG 2	
	I_O_SIM_MOD	00000230	00001117	00000EE8	(3816.)	LONG 2	
	EXPLAIN_DIAGNOSIS_	00001118	00001123	0000000C	(12.)	LONG 2	
	EXPLAIN_DIAGNOSIS	00001124	0000112F	0000000C	(12.)	LONG 2	
	FAULT_ANALYZE_	00001130	0000113B	0000000C	(12.)	LONG 2	
	FAULT_ANALYZE	0000113C	00001147	0000000C	(12.)	LONG 2	
$ZERO		00001200	00001207	00000008	(8.)	PAGE 9	PIC,USR,OVR,REL,LCL, SHR,NOEXE, RD,NOWRT,NOVEC
	TEXT_IO_	00001200	00001207	00000008	(8.)	PAGE 9	
	I_O_SIM_MOD	00001200	00001200	00000001	(1.)	PAGE 9	
LIB$INITIALIZE		00001208	0000123B	00000034	(52.)	LONG 2	NOPIC,USR,CON,REL,GBL,NOSHR,NOEXE, RD,NOWRT,NOVEC
	ADA$ELAB_MAIN_SIM_MOD3	00001208	0000123B	00000034	(52.)	LONG 2	
$DATA		00001400	0000144B	0000004C	(76.)	LONG 2	PIC,USR,CON,REL,LCL,NOSHR,NOEXE, RD, WRT,NOVEC
	Y_OUT_MOD	00001400	00001400	00000001	(1.)	LONG 2	
	TEXT_IO	00001404	00001409	00000006	(6.)	LONG 2	
	I_O_SIM_MOD	0000140C	0000143F	00000034	(52.)	LONG 2	
	EXPLAIN_DIAGNOSIS	00001440	00001440	00000001	(1.)	LONG 2	
	FAULT_ANALYZE	00001444	0000144B	00000008	(8.)	LONG 2	

Psect Name	Module Name	Base	End	Length	Align	Attributes
$CODE		00001600	0000480D	0000320E (12814.)	LONG 2	PIC,USR,CON,REL,LCL, SHR, EXE, RD,NOWRT,NOVEC
	ADA$ELAB_MAIN_SIM_MOD3					
		00001600	0000161D	0000001E (30.)	LONG 2	
	TYP_DEF_SIM_	0000161E	00001654	00000037 (55.)	BYTE 0	
	IO_EXCEPTIONS_	00001655	0000165A	00000006 (6.)	BYTE 0	
	Y_OUT_MOD_	0000165B	00001691	00000037 (55.)	BYTE 0	
	Y_OUT_MOD	00001692	0000179C	0000010B (267.)	BYTE 0	
	TEXT_IO_	0000179D	000017B3	00000017 (23.)	BYTE 0	
	TEXT_IO	000017B4	00001865	000000B2 (178.)	BYTE 0	
	I_O_SIM_MOD_	00001866	00001885	00000020 (32.)	BYTE 0	
	I_O_SIM_MOD	00001886	00004278	000029F3 (10739.)	BYTE 0	
	EXPLAIN_DIAGNOSIS					
		00004279	00004298	00000020 (32.)	BYTE 0	
	EXPLAIN_DIAGNOSIS					
		00004299	00004391	000000F9 (249.)	BYTE 0	
	FAULT_ANALYZE_	00004392	000043B1	00000020 (32.)	BYTE 0	
	FAULT_ANALYZE	000043B2	0000461A	00000269 (617.)	BYTE 0	
	MAIN_SIM_MOD3	0000461B	0000480D	000001F3 (499.)	BYTE 0	

+------------------+
! Symbols By Name !
+------------------+

Symbol	Value	Symbol	Value	Symbol	Value
ADA$ELAB_MAIN_SIM_MOD3	00001600-R				

Key for special characters above:

+--------------------+
! * - Undefined !
! U - Universal !
! R - Relocatable !
! X - External !
! V - Vectored !
+--------------------+

```
                                  +-----------------+
                                  ! Image Synopsis !
                                  +-----------------+

Virtual memory allocated:                  00000200 00004BFF 00004A00 (18944. bytes, 37. pages)
Stack size:                                      20. pages
Image header virtual block limits:                1.         1. (     1. block)
Image binary virtual block limits:                2.        37. (    36. blocks)
Image name and identification:             MAIN_SIM_MOD3 01
Number of files:                                 19.
Number of modules:                               19.
Number of program sections:                      16.
Number of global symbols:                       911.
Number of image sections:                        15.
User transfer address:                     00001600
Debugger transfer address:                 7FFEDF68
Number of code references to shareable images:   24.
OTS transfer address - LIB$INITIALIZE:     0000480E
Image type:                                EXECUTABLE.
Map format:                                DEFAULT in file USER2:[TFL.ADADIR]MAIN_SIM_MOD3.MAP;3
Estimated map length:                      83. blocks
                                +---------------------+
                                ! Link Run Statistics !
                                +---------------------+

Performance Indicators                     Page Faults       CPU Time     Elapsed Time
----------------------                     -----------       --------     ------------
    Command processing:                            145     00:00:00.16     00:00:00.18
    Pass 1:                                        417     00:00:02.33     00:00:04.05
    Allocation/Relocation:                          31     00:00:00.09     00:00:00.46
    Pass 2:                                         63     00:00:01.21     00:00:02.59
    Map data after object module synopsis:           4     00:00:00.10     00:00:00.10
    Symbol table output:                             0     00:00:00.02     00:00:00.14
Total run values:                                  660     00:00:03.91     00:00:07.52

Using a working set limited to 1350 pages and 686 pages of data storage (excluding image)

Total number object records read (both passes):   944
    of which 41 were in libraries and 211 were DEBUG data records containing 26139 bytes
2835 bytes of DEBUG data were written,starting at VBN 38 with 6 blocks allocated

Number of modules extracted explicitly             = 0
    with 2 extracted to resolve undefined symbols

0 library searches were for symbols not in the library searched

A total of 0 global symbol table records was written

LINK/MAP=[]MAIN_SIM_MOD3/EXE=[]MAIN_SIM_MOD3 SYS$INPUT:/OPTIONS
```